Drum Machine

Computer Engineering Senior Project Report

Ryan Frawley
Advisor: Andrew Danowitz

California Polytechnic State University, San Luis Obispo
June 2017

Table of Contents

Introduction

Project Overview

The drum machine is a step sequencer which allows drum loops to be created. The system allows the user to adjust the tempo from 30 to 340 beats per minute. There are 14 different drum samples available for use in the loop. There are 16 different positions representing time slots in a measure that a drum beat can be placed in by the user. The position of the beats is controlled by pressing any one of 16 buttons on a pad consisting of four rows and four columns to make 16 buttons in total. All drum loops created on the drum machine are in the 4/4 time signature.

Project Outcome and Deliverables

At the end of this project, I created a step sequencer that allows musicians to create simple drum loops in 4/4 time signature. The drum machine is a good choice for people who do not have access to a drumset or software which can perform a similar task. It is easy to use and offers a wide variety of different sounds to suit many different music genres.

Functional Requirements

Early in the design phase of the drum machine, the following requirements were established. The requirements were inspired by early programmable digital drum machines like the Roland TR-808 and TR-909.

1. The drum machine must have at least 10 different drum samples.
2. The tempo must be adjustable up to 340 beats per minute.
3. A loop must be represented by a light sweeping across a 4x4 button pad.
4. The tempo and drum must be adjustable during playback of a loop.
5. Information about the tempo and drum must be displayed on an LCD.
6. Audio must be played through a 3.5mm jack.
7. The hardware must fit in an enclosure no larger than 7”x5”x3”.

8. The drum machine must consume less than 12.5 W at any time.

System Architecture

Hardware

The two main components in the drum machine are a Raspberry Pi 3 and an Arduino Uno. The Raspberry Pi is connected to a 4x4 button pad using four wires that provide 5V and ground to power the pad as well as the data and clock lines needed to communicate with it. The Pi also outputs audio to speakers through a 3.5mm cable. The Arduino is connected to two potentiometers used to control the tempo and desired drum, and a 16x2 LCD to display information about the tempo and drum. The Raspberry Pi sends and receives data from the button pad using I2C. Each button in the pad has its own address which can be used to read or write values to the button. For instance, when a button is pressed, the Raspberry Pi reads from that button to detect whether that button is on or off. Similarly, after the button press the Pi sends a command to the button pad to either turn on or turn off the LED behind that button.

The Arduino reads the voltage output by each potentiometer through its analog input pins, converts it to a digital value with its onboard ADC, and then sends that value to the Raspberry Pi using serial communication through the USB cable connecting the two. It also writes the value read from each potentiometer to the LCD to give the user useful information about the current loop. The LCD is used in nibble mode, so it only uses four the the eight data pins on the LCD. The hardware block diagram is shown in figure 1. A more detailed schematic is shown below that in figure 2.

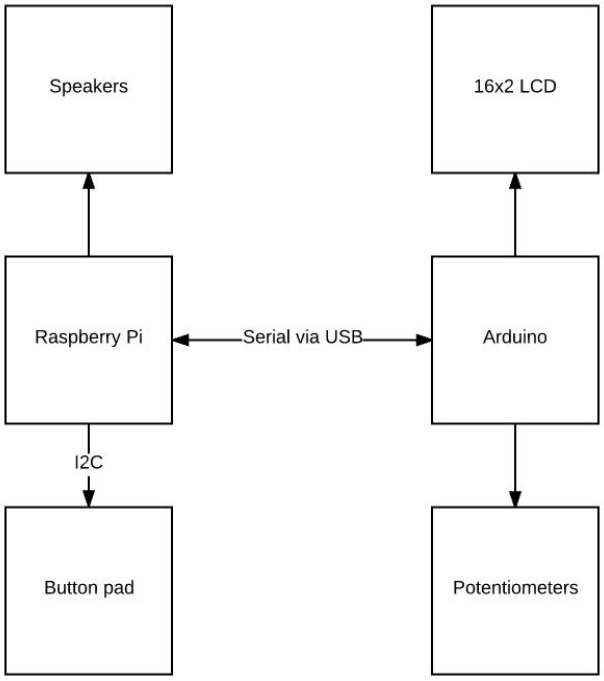

Figure 1: Hardware Block Diagram

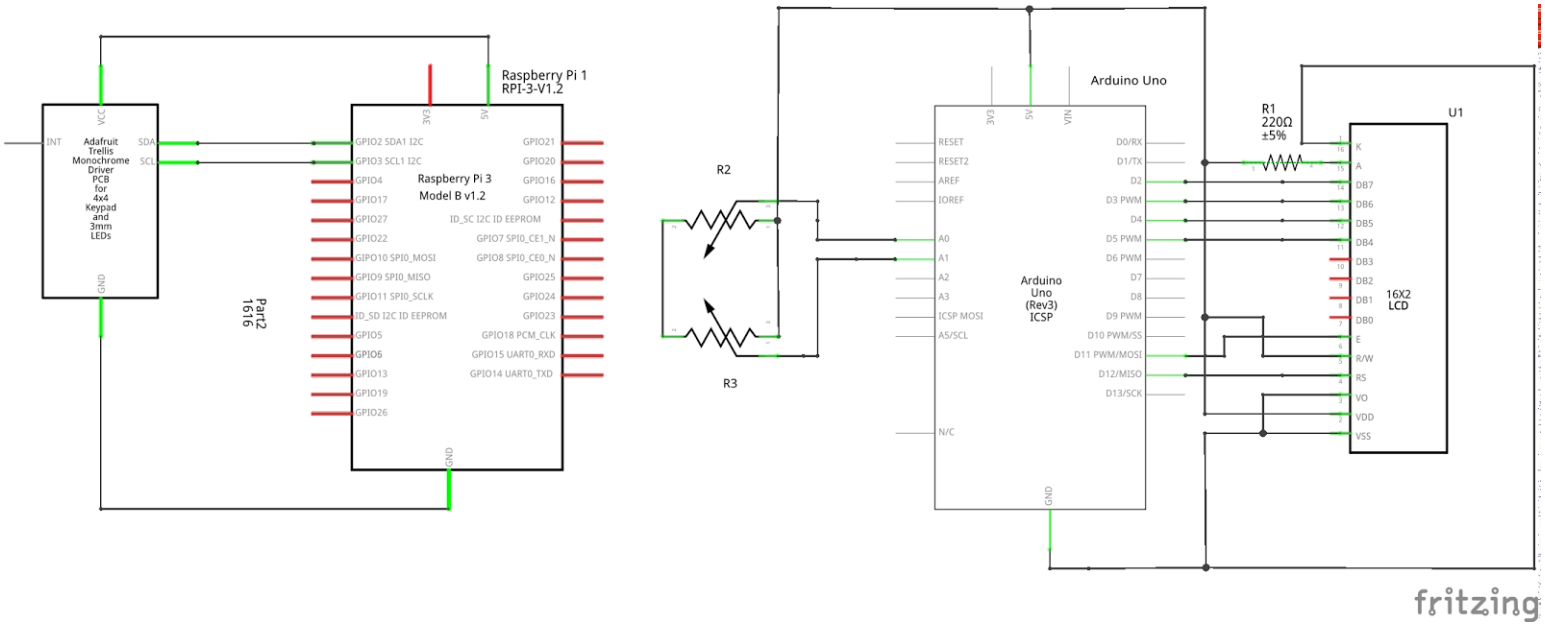

Figure 2: System Schematic

The hardware is contained within an aluminum project enclosure that I designed. In order to create the cutouts for all of the components, I used LibreCAD to create CAD

files with the location and dimensions of each component that I needed mounted. I then took the CAD files and the aluminum enclosure to a machine shop where I had it cut with a CNC abrasive waterjet cutter. The drawings for the top and side of the enclosure are shown in figure 3.

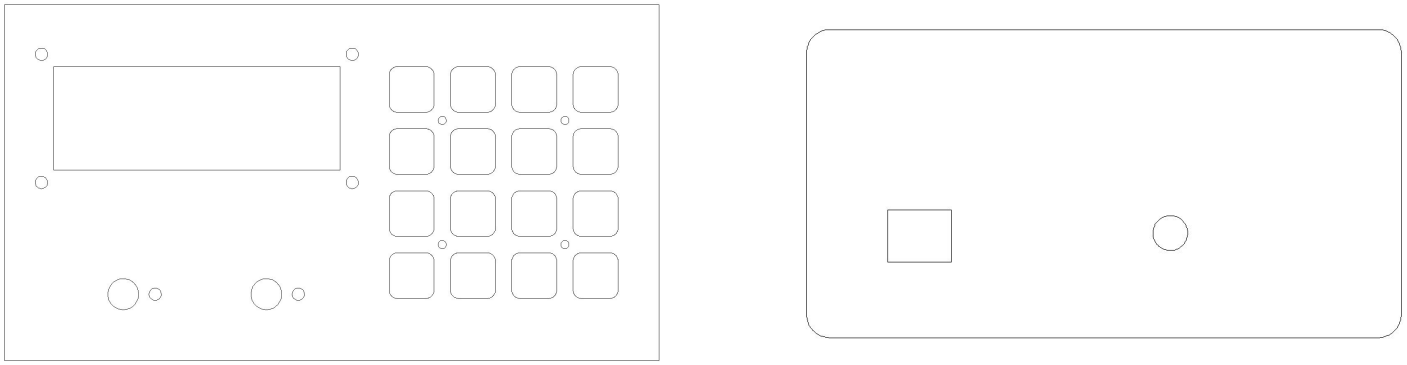

Figure 3: Enclosure CAD Drawings

After I received the machined parts, I mounted the potentiometers, LCD, and button pad to the top piece and the 3.5mm jack and micro USB cable to the side piece. The USB cable mounted to the side is used to power the drum machine. Space inside of the enclosure was very limited, so I let the arduino stick out to the side to avoid damaging any cables inside of the enclosure. All components were mounted using nylon standoffs to prevent short circuits on the aluminum enclosure. The final result with all components mounted is shown in figure 4.

Figure 4: Assembled drum machine

Software

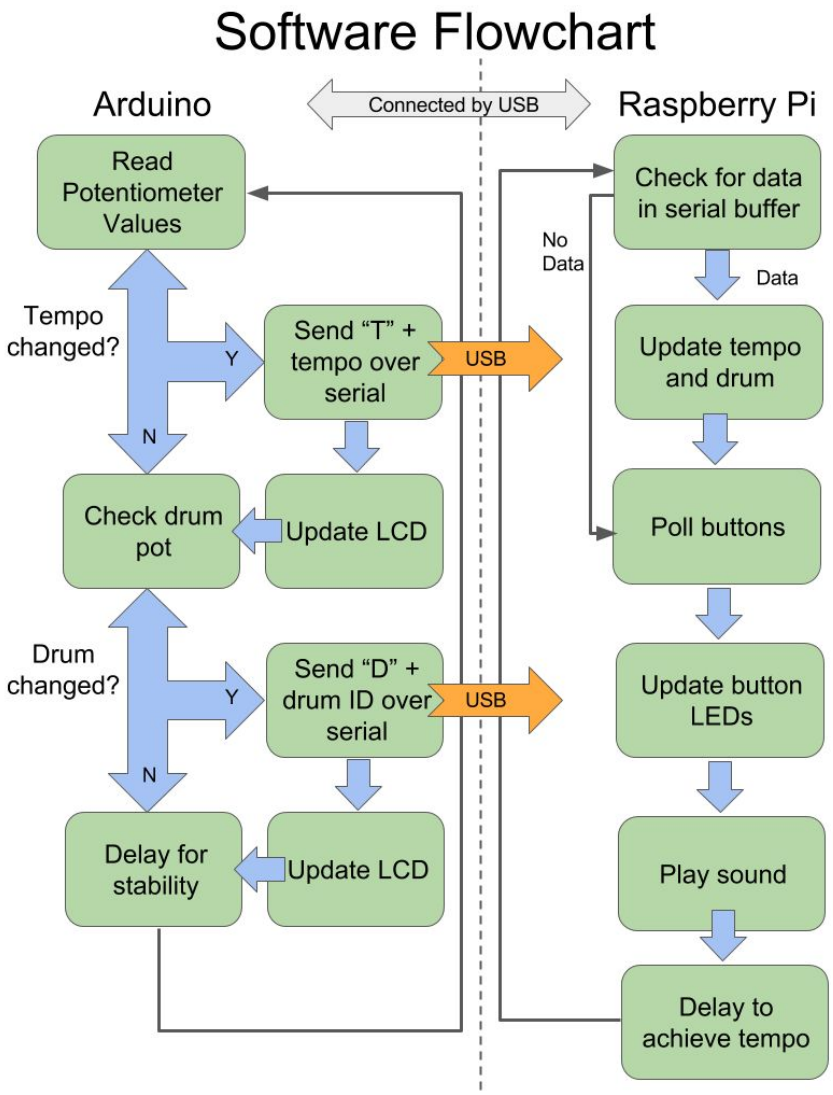

Figure 5: Software Flowchart

Raspberry Pi 3

The Raspberry Pi contains software written in Python. The software running on the Pi controls the button pad and the LEDs behind each button using the Adafruit Trellis library. It also controls the playback of drum samples. If any data is sent from the Arduino, the Pi reads that data and updates either the current drum or the tempo depending on the prefix of the data. If the data starts with a 'T', then the Pi updates the current tempo. If the data starts with a 'D', the Pi updates the drum sample that is currently selected. This is shown in the code below.

```
# If we got a 'T', it's a tempo. Update the tempo
if (temp.strip()=='T'):
    temp = ser.readline().decode('utf-8')
    if temp.strip().isdigit():
        if (int(temp.strip()) > 30):
            tempo = 60 / (int(temp.strip()))
# If we got a 'D', it's a drum ID. Update the drum.
if (temp.strip()=='D'):
   inst = ser.readline().decode('utf-8')
   if inst.strip().isdigit():
        cur = selectInstrument(inst, i)
```

Audio playback is provided through Pygame's audio library. All drum samples are stored as .ogg files sampled at 44.1 kHz with bitrates ranging from 80 kbits/s to 112 kbit/s. There are 14 different drum samples and one additional empty one to be used for beats with no drum on them. This is the initial state for every beat.

After initializing all of the sounds, serial communication, and the button pad, the software enters an endless loop that contains the main logic of the drum machine. At the start of every loop, the software checks to see if there is any data in the serial buffer. If there is, it checks to see if there are more than 20 bytes in the buffer. If there are more than 20 bytes, the buffer is cleared. This is done in order to improve the responsiveness of the system. If the buffer isn't cleared, old tempos or drums can clog up the buffer and delay values being sent from the Arduino. After this, a new value is read from the serial input. The software updates either the drum or the tempo and then it polls all of the buttons to see if any have been pressed since the last loop iteration. If there are any changes, the array containing all of the samples is updated and the LED behind that button is toggled. Next, the software toggles the LED behind the button representing the current beat, plays the sample in that spot, and then toggles the LED again. This allows the drum machine to produce a sweeping motion to show which beat the loop is currently on.

Arduino Uno

The software on the Arduino was written in C++. The main purpose of the software on the Arduino is to read potentiometer values, print to the LCD, and send potentiometer values to the Raspberry Pi through serial. The code initializes the LCD to use nibble mode and the serial port to use the USB port at 9600 baud. It then enters the main loop where it reads the potentiometer values and performs an action based on

those values. For the tempo potentiometer, the code only updates the tempo if the potentiometer value has changed more than 2. This is to provide more stable reads as the value wavers slightly. If there is a change, then the LCD is updated and the new value is sent to the Pi using serial communication. Next, the code checks the drum potentiometer value to see if it has changed enough to move into the next sample. The maximum value read from each potentiometer is 1024 and it is divided by the number of samples to produce a range of values for each sample. In this implementation, there are 14 samples so each sample has a potentiometer value range of $\frac{1024}{14} = 73$. Again, if the value changes, the LCD is updated and the new value is sent to the Raspberry Pi.

Challenges

One of the most significant challenges encountered during the implementation the drum machine was finding a way to get the Raspberry Pi and the Arduino to communicate reliably. Initially, I planned on using the Raspberry Pi's GPIO pins to communicate with the Arduino using UART. This would have required the use of a logic level converter because the Arduino outputs 5V and the Raspberry Pi has a max input voltage of 3.3V. I was also planning on powering the Arduino straight from the 5V output of the Raspberry Pi. This would have required the use of four cables to connect the Arduino and the Raspberry Pi. Using a USB cable allowed me to provide both power and communication between the two boards using only a single cable.

The first version of the software controls for serial communication produced unsatisfactory results. Tempo and drum values that were sent from the Arduino were frequently dropped or delayed significantly by the time they were processed on the Raspberry Pi. This led to the drum machine being unresponsive to user changes when adjusting the tempo or desired drum. I fixed the problem by modifying the software on the Arduino so that values were only sent to the Pi if they were changed. On the Raspberry Pi, I modified the software so that only 20 bytes of data were held in the serial input buffer at any time. This improved the responsiveness of the system significantly.

Once the hardware and software components were complete, I mounted all of the hardware in a custom aluminum project enclosure that I designed CAD files for. The physical space inside of the enclosure was extremely limited with all of the hardware and cables inside of it. Arranging all of the hardware in a way that it would fit neatly

inside of the enclosure was a significant challenge. I bought a mini breadboard that was small enough to fit comfortably inside of the enclosure while still leaving room for the other components. I also had to buy a new USB cable for the Arduino that had a 90 degree bend so it would fit inside the enclosure. In the end, I left the Arduino sticking out of the enclosure by about two inches because I did not want to risk breaking any cables which were already stressed by the bends I had to make to fit the rest of the hardware. This problem could have been mitigated by using a custom printed circuit board in order to limit the number of wires inside of the enclosure.

Future Improvements

In the future, I'd like to expand on the drum machine by adding more complex features. I'd like to be able to support more time signatures than 4/4. In order to accomplish this, I could expand the number of buttons on the pad. Using four of the 4x4 pads would provide a total of 64 buttons to work with which would greatly improve the amount of control the user has over the drum loop. Adding more buttons would also allow users to create loops with things like syncopation which are important in genres such as Jazz.

Currently, the drum machine only supports one drum on each beat of the loop. In the future, I would like to improve this by allowing users to layer multiple drum beats on top of each other on each beat. Pygame's audio library supports this through the use of channels. It has support for up to eight channels so eight different sounds can play simultaneously. This would allow users to create more complex drum loops that more closely represent drums that are actually used in songs. Having at least four channels would be ideal as this could mimic a human playing the drums with two hands and two feet available for use on different drums.

Improving the audio quality is very important if the drum machine is going to be used for any recording purposes. The built in 3.5mm jack on the Raspberry Pi 3 produces a lot of static noise. I was able to reduce the noise slightly by turning off audio dithering, but there was still a significant hiss noticeable. Using a USB sound card or an external DAC would likely solve this problem. I'd also like to add support for using the Pi's built in Bluetooth to wirelessly connect to speakers. This would require either a software interface or an additional portion of the software which would search and connect to

Bluetooth devices when a button was pressed. The aluminum enclosure may block some of the wireless signals, so the material might have to be changed.

Conclusion

The drum machine fills all of the requirements created at the beginning of its design. It is simple to use and anyone can make their own drum loops in a matter of minutes. The aluminum project enclosure ensures that the drum machine is sturdy enough to handle moderate use. In the future, the reliability and compactness of the drum machine could be improved by using custom printed circuit boards to connect all of the components. Despite the room for future improvements, I am happy with the final product. I look forward to seeing how the drum machine will be used by others.

Appendix

Raspberry Pi Python Code

```
#-------------------------------------------------------------------------------
# Title: Drum machine Raspberry Pi code
# Description: Controls the button pad, plays back drum samples, and reads
#  tempo and drum values sent from the Arduino.
# Author: Ryan Frawley
# Quarter: Spring 2017
#-------------------------------------------------------------------------------

import time
import Adafruit_Trellis
import pygame
import serial

# The array that holds drum samples for each button in the 4x4 pad.
A = [0] * 16

pygame.mixer.init(buffer=512)

# Value used to make the buttons act as toggle switches
LATCHING = 1
# Set the mode here:
MODE = LATCHING

# Initialize the buttons
matrix0 = Adafruit_Trellis.Adafruit_Trellis()
trellis = Adafruit_Trellis.Adafruit_TrellisSet(matrix0)
NUMTRELLIS = 1

# Set the number of keys. Adjustable for any possible expansions with larger
# button pads.
numKeys = NUMTRELLIS * 16

# Set this to the number of the I2C bus that the Trellises are attached to:
I2C_BUS = 1

# Tell the Pi to read and write values to address 0x70, the address of the
# button pad.
trellis.begin((0x70, I2C_BUS))

# Initialize serial communications to read from the USB port using 9600 baud
# in non-blocking mode
ser = serial.Serial('/dev/ttyACM0', 9600, timeout = 0)

# Load in all of the drum samples.
empty = pygame.mixer.Sound("/home/pi/drummachine/samples/none.ogg")
kick =  pygame.mixer.Sound("/home/pi/drummachine/samples/kick.ogg")
snare = pygame.mixer.Sound("/home/pi/drummachine/samples/snare.ogg")
hihat = pygame.mixer.Sound("/home/pi/drummachine/samples/hihat.ogg")
tomtom = pygame.mixer.Sound("/home/pi/drummachine/samples/tom.ogg")
clap = pygame.mixer.Sound("/home/pi/drummachine/samples/clap.ogg")
ride = pygame.mixer.Sound("/home/pi/drummachine/samples/ride.ogg")
```

```
agogo = pygame.mixer.Sound("/home/pi/drummachine/samples/agogo.ogg")
cowbell = pygame.mixer.Sound("/home/pi/drummachine/samples/cowbell.ogg")
kickGrit = pygame.mixer.Sound("/home/pi/drummachine/samples/kick-gritty.ogg")
rim = pygame.mixer.Sound("/home/pi/drummachine/samples/nasty_rim.ogg")
perc = pygame.mixer.Sound("/home/pi/drummachine/samples/perc.ogg")
shaker = pygame.mixer.Sound("/home/pi/drummachine/samples/shaker.ogg")
snap = pygame.mixer.Sound("/home/pi/drummachine/samples/snap.ogg")
snareDist = pygame.mixer.Sound("/home/pi/drummachine/samples/snare-dist.ogg")


# Set all of the entries in our array to no sound.
def initialize():
    for i in range(numKeys):
        A[i] = empty


# Use the value sent from the Arduino to update the current drum.
def selectInstrument(inst, i):
    cur = kick

    if inst.strip().isdigit():
        ins = int(inst.strip())
        if (ins == 1):
            cur = agogo
        elif (ins == 2):
            cur = clap
        elif (ins == 3):
            cur = cowbell
        elif (ins == 4):
            cur = hihat
        elif (ins == 5):
            cur = kick
        elif (ins == 6):
            cur = kickGrit
        elif (ins == 7):
            cur = rim
        elif (ins == 8):
            cur = perc
        elif (ins == 9):
            cur = ride
        elif (ins == 10):
            cur = shaker
        elif (ins == 11):
            cur = snap
        elif (ins == 12):
            cur = snare
        elif (ins == 13):
            cur = snareDist
        elif (ins == 14):
            cur = tomtom

    return cur

# Poll all buttons to see if there are any changes
def checkInput(cur):
    if trellis.readSwitches():
        # go through every button
        for i in range(numKeys):
            # if it was pressed...
            if trellis.justPressed(i):
```

```
                    # Alternate the LED
                    if trellis.isLED(i):
                        A[i] = empty
                        trellis.clrLED(i)
                    else:
                        A[i] = cur
                        trellis.setLED(i)
    # tell the trellis to set the LEDs we requested
    trellis.writeDisplay()

def main():
    tempo = 120         # Initialize our tempo to 120 BPM
    tempo = 60/tempo    # Calculate the delay needed to get our desired tempo
    ind = 0
    cur = kick
    initialize()
    while 1:
        for i in range(numKeys):
          # If there's any data in the serial buffer, read that data.
          if ser.in_waiting > 0:
                # If there's more than 20 bytes in the buffer, clear it.
                if ser.in_waiting >= 20:
                     ser.reset_input_buffer()
                temp = ser.readline().decode('utf-8')

                # Check to see what the prefix of the data is
                if (temp.strip().isalpha()):
                     # If we got a 'T', it's a tempo. Update the tempo
                     if (temp.strip()=='T'):
                          temp = ser.readline().decode('utf-8')
                          if temp.strip().isdigit():
                                if (int(temp.strip()) > 30):
                                     tempo = 60 / (int(temp.strip()))
                     # If we got a 'D', it's a drum ID. Update the drum.
                     if (temp.strip()=='D'):
                          inst = ser.readline().decode('utf-8')
                          if inst.strip().isdigit():
                                cur = selectInstrument(inst, i)

          checkInput(cur)
          # Toggle the LED on the current button
          if trellis.isLED(i):
              trellis.clrLED(i)
          else:
              trellis.setLED(i)
          trellis.writeDisplay()
          # Play the drum sample for the current beat
          pygame.mixer.Sound.play(A[i])
          # Delay to achieve the desired tempo
          time.sleep(tempo)
          # Toggle the LED again
          if trellis.isLED(i):
              trellis.clrLED(i)
          else:
              trellis.setLED(i)
          trellis.writeDisplay()

if __name__ == "__main__":
     main()
```

Arduino C++ Code

```
/* ---------------------------------------------------------------------------
 * Title: Drum machine arduino code.
 * Description: Reads potentiometer values, writes to the LCD, and sends drum
 *  and tempo values to the Raspberry Pi through serial.
 * Author: Ryan Frawley
 * Quarter: Spring 2017
 * ---------------------------------------------------------------------------
*/
#include <LiquidCrystal.h>

#define MAX_POT 1024           // The max digital value read from the pots
#define N_DRUMS 14             // Number of drum samples
#define DIV MAX_POT / N_DRUMS  // Width of each sample's pot value.

// initialize the LCD in nibble mode
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// Initialize serial communication and the LCD
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
   // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
}

int flag = 0;
int temp = 0;
// Returns the opposite value of the current flag value
int flipFlag(int flag) {
  flag = !flag;
  return flag;
}
// Update the drum name displayed on the LCD and send the new ID to the Pi.
int changeDrum(const char *name, int id) {
  lcd.setCursor(0, 1);
  lcd.print("                ");   // Clear the bottom line of the LCD
  lcd.setCursor(0, 1);
  lcd.print(name);                 // Print the name of the drum
  Serial.println("D");             // Send 'D' followed by the drum ID to the Pi
  Serial.println(id);
}
/* Reads the value from the potentiometer and updates it if it has changed since
   the last read */
void checkPot(const char *name, int val, int drumPotVal) {
  /* If the value has changed enough to be in another drum ID range, update the
     value */
  if (drumPotVal >= DIV * (val-1) && drumPotVal < DIV * val) {
    if (flag != val) {
      changeDrum(name, val);
      flag = val;
    }
  }
}
// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0 and pin 1 (potentiometers)
```

```
  int tempoPotVal = analogRead(A0);
  int drumPotVal = analogRead(A1);
  /* If the tempo pot value has changed more than 2, update the tempo. This is
     done for read stability */
  if (temp < tempoPotVal - 2 || temp > tempoPotVal + 2) {
    Serial.println("T");
    Serial.println(tempoPotVal / 3); // Divide the pot value by 3 to get tempo
    lcd.setCursor(0, 0);
    lcd.print("                ");   // Clear top line of LCD
    lcd.setCursor(0, 0);
    lcd.print("Tempo: ");
    lcd.print(tempoPotVal / 3);      // Print tempo to LCD
    lcd.print(" BPM");
    lcd.setCursor(0, 1);
    temp = tempoPotVal;
  }

  // Check the drum potentiometer's value to see which sample is selected
  checkPot("Agogo", 1, drumPotVal);
  checkPot("Clap", 2, drumPotVal);
  checkPot("Cowbell", 3, drumPotVal);
  checkPot("Hi-hat", 4, drumPotVal);
  checkPot("Kick", 5, drumPotVal);
  checkPot("Kick (gritty)", 6, drumPotVal);
  checkPot("Nasty Rim", 7, drumPotVal);
  checkPot("Percussion Block", 8, drumPotVal);
  checkPot("Ride cymbal", 9, drumPotVal);
  checkPot("Shaker", 10, drumPotVal);
  checkPot("Snap", 11, drumPotVal);
  checkPot("Snare", 12, drumPotVal);
  checkPot("Snare (distort)", 13, drumPotVal);
  checkPot("Tom Tom", 14, drumPotVal);

  delay(100);        // delay in between reads for stability
}
```

Bill of Materials

Part Number	Part Name	Unit Cost	Quantity	Cost
1	Arduino Uno	$18.82	1	$18.82
2	Raspberry Pi 3 Model B	$35.97	1	$35.97
3	32 GB Micro SD card	$9.99	1	$9.99
4	3mm blue LED	$0.04	16	$0.64
5	Adafruit Trellis PCB	$9.95	1	$9.95
6	Silicone Elastomer 4x4 Button Keypad	$4.95	1	$4.95
7	16x2 LCD	$10.95	1	$10.95
8	10kΩ potentiometer	$0.95	2	$1.90
9	Potentiometer knob	$0.50	2	$1.00
10	3.5mm male to male cable	$2.50	1	$2.50
11	Mini breadboard	$5.49	1	$5.49
12	Right angle USB A to USB B cable	$6.99	1	$6.99
13	M2/M3 standoffs, screws, and bolts kit	$11.96	1	$11.96
14	Panel mount 3.5mm male to female cable	$6.55	1	$6.55
15	BOX3 1455N Black Aluminum Box	$22.55	1	$22.55
16	220Ω Resistor	$0.05	1	$0.05
17	Waterjet Cutting Fee	$80.00	1	$80.00
Total Cost				$230.26

Table 1: Bill of Materials